C- Programming On Embedded System

Pointers and malloc

Girish S Kumar

What are differences between malloc and array

int array[n] ;
Allocates space for n integers on the stack, which is usually pretty small. Using memory on the stack is much faster than the alternative, but it is quite small and it is easy to overflow the stack (i.e. allocate too much memory) if you do things like allocate huge arrays or do recursion too deeply. You do not have to manually deallocate memory allocated this way, it is done by the compiler when the array goes out of scope.

malloc on the other hand allocates space in the heap, which is usually very large compared to the stack. You will have to allocate a much larger amount of memory on the heap to exhaust it, but it is a lot slower to allocate memory on the heap than it is on the stack, and you must deallocate it manually via free when you are done using it.

Pointers

Pointers in C language is a variable that stores/points the address of another variable. A Pointer in C is used to allocate memory dynamically i.e. at run time. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

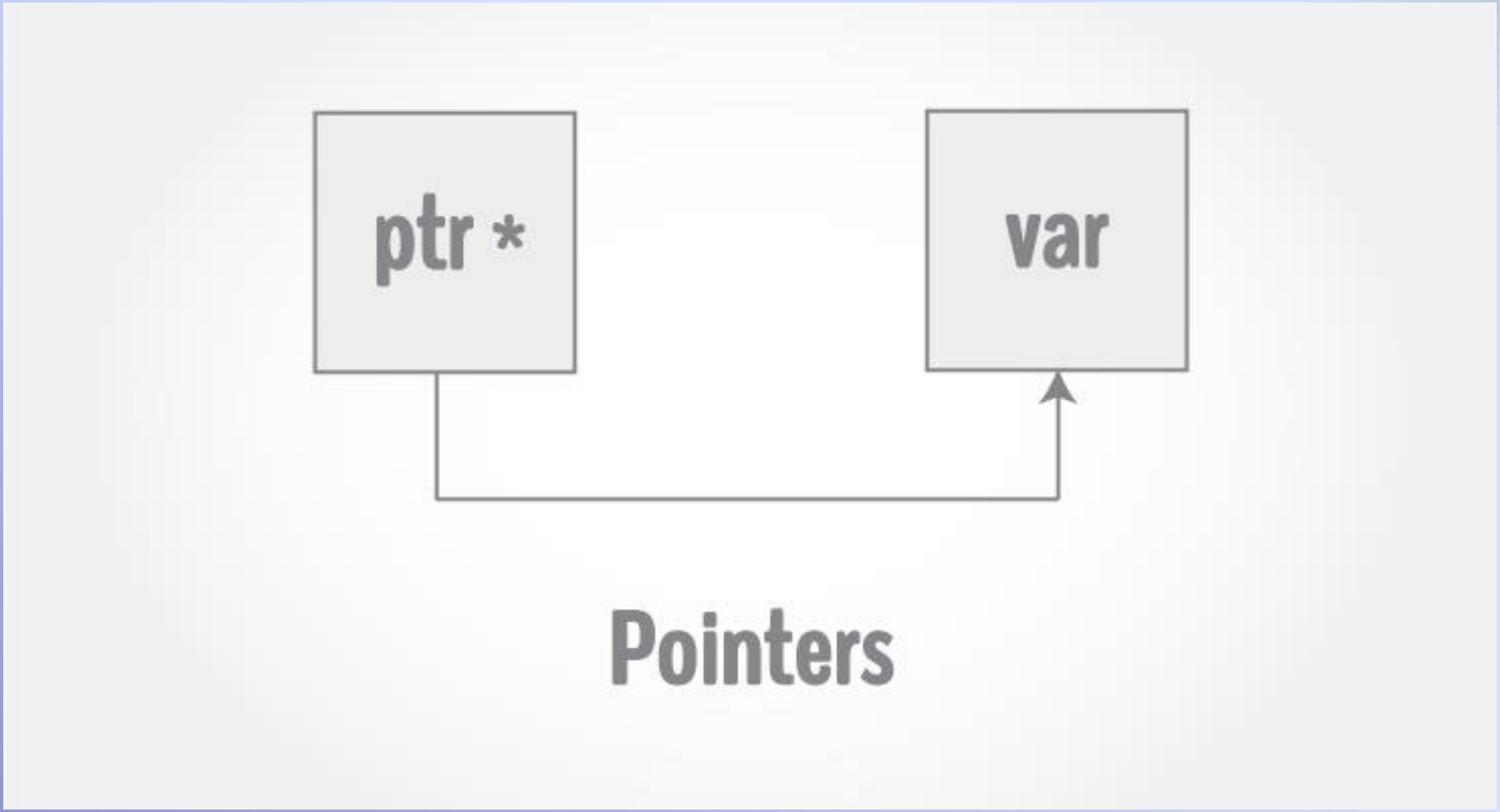
- Normal variable stores the value whereas pointer variable stores the address of the variable.
- The content of the C pointer **always be a whole number** i.e. address.
- Always C pointer is initialized to null, i.e. int *p = null.
- The value of null pointer is 0.
- & symbol is used to get the address of the variable.
- * symbol is used to get the value of the variable that the pointer is pointing to.
- If a pointer in C is assigned to NULL, it means it is pointing to nothing.
- Two pointers can be subtracted to know how many elements are available between these two pointers.
- But, Pointer addition, multiplication, division are not allowed.
- Question : What is the size of any pointer is 2 byte :

What are pointers

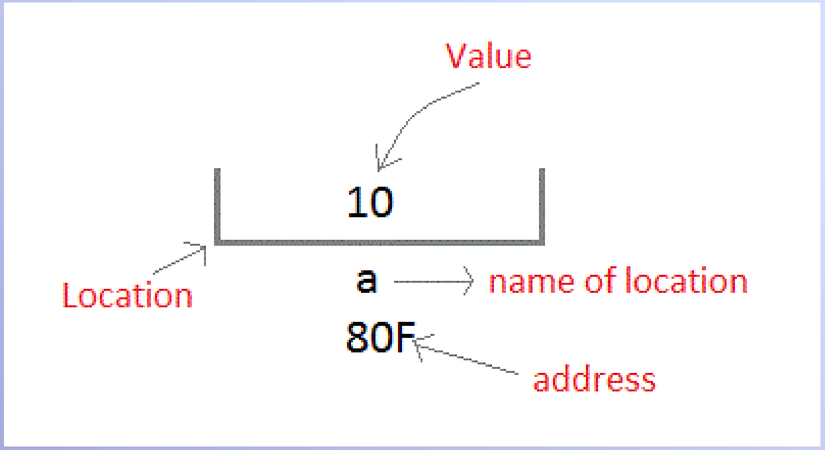
In C, there is a special variable that stores just the address of another variable. It is called Pointer variable or, simply, a pointer.

<u>Declaration of Pointer</u>

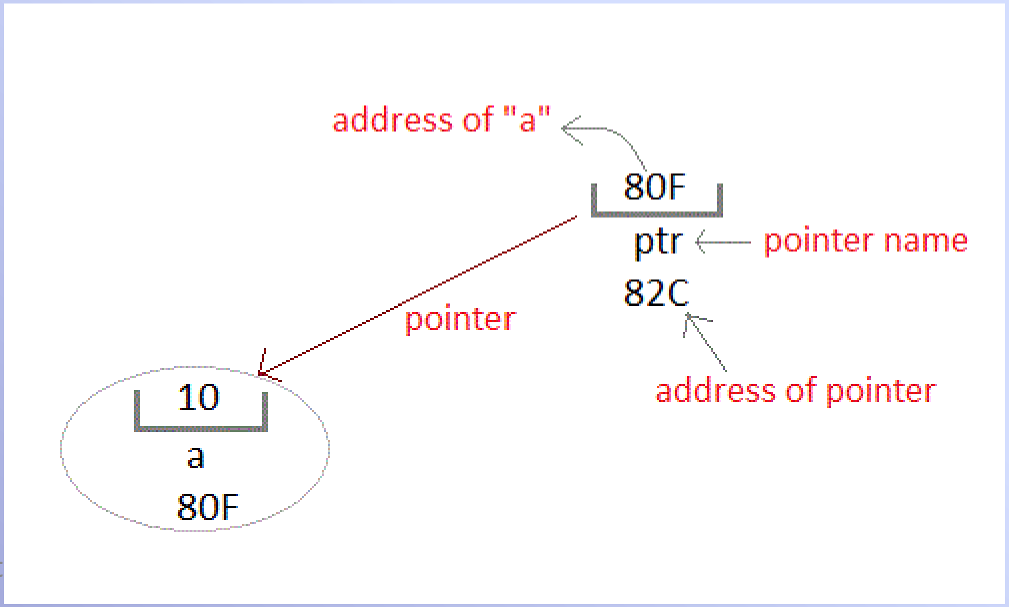
```
data_type* pointer_variable_name;
int* p;
```


ptr *
var
Pointers

int a = 10



Whenever a **variable** is declared in the program, system allocates a location i.e an address to that variable in the memory, to hold the assigned value. This location has its own address number. Let us assume that system has allocated memory location 80F for a variable **a**. We can access the value 10 by either using the variable name **a** or the address 80F. Since the memory addresses are simply numbers they can be assigned to some other variable. The variable that holds memory address are called **pointer variables**. A **pointer** variable is therefore nothing but a variable that contains an address, which is a location of another variable. Value of **pointer variable** will be stored in another memory location.



Reference operator (&) and Dereference operator (*)

& is called reference operator. It gives you the address of a variable.
* is called the dereference operator . Itgives you the value from the address

Note: The * sign when declaring a pointer is not a dereference operator. It is just a similar notation that creates a pointer.

Mistakes that happens with pointer

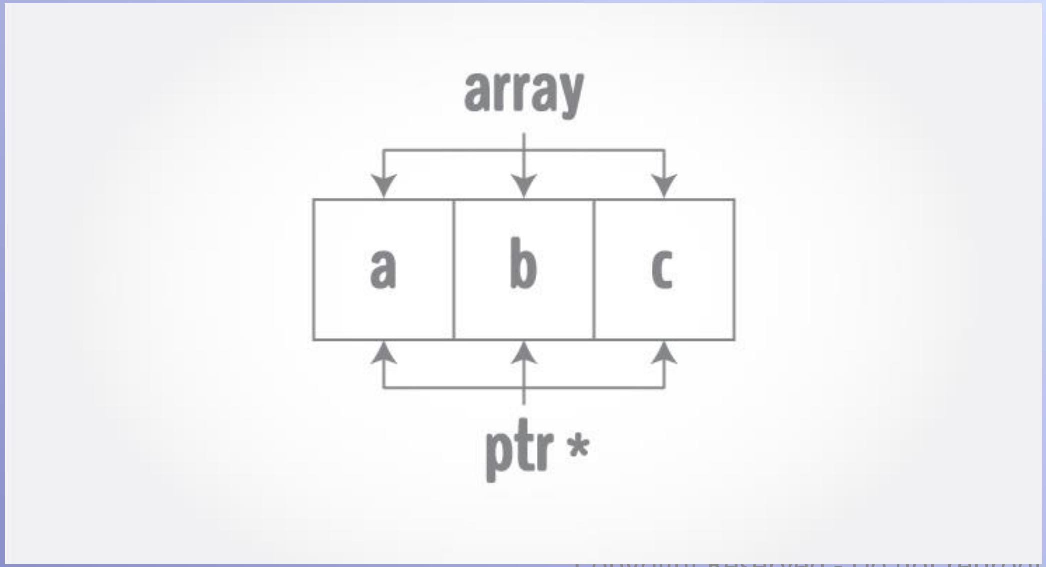
```
int c, *pc;

// Wrong! pc is address whereas, c is not an address.
pc = c;

// Wrong! *pc is the value pointed by address whereas, %amp;c is an address.
*pc = &c;

// Correct! pc is an address and, %amp;pc is also an address.
pc = &c;

// Correct! *pc is the value pointed by address and, c is also a value.
*pc = c;
```

Example

```
#include <stdio.h>
int main()
{
  int *ptr, q;
  q = 50;
  /* address of q is assigned to ptr */
  ptr = &q;
  /* display q's value using ptr variable */
  printf("%d", *ptr);
  return 0;
}
```

Relation between Pointers and arrays

```
#include <stdio.h>
int main()
{
  char charArr[4];
  int i;
  for(i = 0; i < 4; ++i)
  {
    printf("Address of charArr[%d] = %u\n", i, &charArr[i]);
  }
  return 0;
}
```



In C programming, name of the array always points to address of the first element of an array.

In the above example, arr and &arr[0] points to the address of the first element.

&arr[0] is equivalent to arr
Since, the addresses of both are the same, the values of arr and &arr[0] are also the same.

arr[0] is equivalent to *arr (value of an address of the pointer)
Similarly,

&arr[1] is equivalent to (arr + 1) AND, arr[1] is equivalent to *(arr + 1).
&arr[2] is equivalent to (arr + 2) AND, arr[2] is equivalent to *(arr + 2).
&arr[3] is equivalent to (arr + 3) AND, arr[3] is equivalent to *(arr + 3).
.
.
&arr[i] is equivalent to (arr + i) AND, arr[i] is equivalent to *(arr + i).

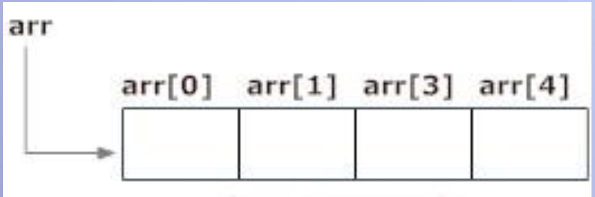


Figure: Array as Pointer

Benefits of Pointers

- Pointers are more efficient in handling Array and Structure.
- Pointer allows references to function and thereby helps in passing of function as arguments to other function.
- It reduces length of the program and its execution time.
- It allows C to support dynamic memory management.